Abstract

The problem of solution transfer between meshes arises frequently in computational physics, e.g. in Lagrangian methods where remeshing occurs. The interpolation process must be conservative, i.e. it must conserve physical properties, such as mass. We extend previous works — which described the solution transfer process for straight sided unstructured meshes — by considering high-order isoparametric meshes with curved elements. The implementation is highly reliant on accurate computational geometry routines for evaluating points on and intersecting Bézier curves and triangles.

Keywords: Remapping, Curved Meshes, Lagrangian, Solution Transfer, Numerical analysis

Contents

1 Introduction **1**
1.1 Overview . . . 2

2 Preliminaries **2**
2.1 General Notation . . . 2
2.2 Bézier Curves . . . 2
2.3 Bézier Triangles . . . 2
2.4 Curved Elements . . . 4
2.5 Shape Functions . . . 4
2.6 Curved Polygons . . . 5

3 Bézier Intersection Problems **6**
3.1 Intersecting Bézier Curves . . . 6
3.2 Intersecting Bézier Triangles . . . 8
3.2.1 Example . . . 8
3.3 Bézier Triangle Inverse . . . 10

4 Solution Transfer **11**

References **11**

1 Introduction

The first part is a general-purpose tool for computational physics problems. The tool enables solution transfer across two curved meshes. Since the tool requires a significant amount of computational geometry, the second half focuses on computational geometry. In particular, it considers cases where the geometric methods used have seriously degraded accuracy due to ill-conditioning.

In computational physics, the problem of solution transfer between meshes occurs in several applications. For example, by allowing the underyling computational domain to change during a simulation, computational effort can be focused dynamically to resolve sensitive features of a numerical solution. Mesh adaptivity (see, for example, [BR78, PVMZ87, PUdOG01]), this in-flight change in the mesh, requires translating the numerical solution from the old mesh to the new, i.e. solution transfer. As another example, Lagrangian or particle-based methods treat each node in the mesh as a particle and so with each timestep the mesh travels *with* the fluid (see, for example, [HAC74]). However, over (typically limited) time the mesh becomes distorted and suffers a loss in element quality which causes catastrophic loss in the accuracy of computation. To overcome this, the domain must be remeshed or rezoned and the solution must be transferred (remapped) onto the new mesh configuration.

When pointwise interpolation is used to transfer a solution, quantities with physical meaning (e.g. mass, concentration, energy) may not be conserved. To address this, there have been many explorations (for example, [JH04, FPP+09, FM11]) of *conservative interpolation* (typically using Galerkin or L_2-minimizing methods). In this work, the author introduces a conservative interpolation method for solution transfer

between high-order meshes. These high-order meshes are typically curved, but not necessarily all elements or at all timesteps.

The existing work on solution transfer has considered straight sided meshes, which use shape functions that have degree $p = 1$ to represent solutions on each element or so-called superparametric elements (i.e. a linear mesh with degree $p > 1$ shape functions on a regular grid of points). However, both to allow for greater geometric flexibility and for high order of convergence, this work will consider the case of curved isoparametric[1] meshes. Allowing curved geometries is useful since many practical problems involve geometries that change over time, such as flapping flight or fluid-structure interactions. In addition, high-order CFD methods ([WFA+13]) have the ability to produce highly accurate solutions with low dissipation and low dispersion error.

1.1 Overview

This work is organized as follows. Section 2 establishes common notation and reviews basic results relevant to the topics at hand. Section 3 is an in-depth discussion of the computational geometry methods needed to implement to enable solution transfer. Section 4 describes the solution transfer process and gives results of some numerical experiments confirming the rate of convergence.

2 Preliminaries

2.1 General Notation

We'll refer to $\mathbf{R}$ for the reals, $\mathcal{U}$ represents the unit triangle (or unit simplex) in $\mathbf{R}^2$: $\mathcal{U} = \{(s, t) \mid 0 \leq s, t, s + t \leq 1\}$. When dealing with sequences with multiple indices, e.g. $s_{m,n} = m + n$, we'll use bold symbols to represent a multi-index: $\boldsymbol{i} = (m, n)$. We'll use $|\boldsymbol{i}|$ to represent the sum of the components in a multi-index. The binomial coefficient $\binom{n}{k}$ is equal to $\frac{n!}{k!(n-k)!}$ and the trinomial coefficient $\binom{n}{i,j,k}$ is equal to $\frac{n!}{i!j!k!}$ (where $i + j + k = n$). The notation δ_{ij} represents the Kronecker delta, a value which is 1 when $i = j$ and 0 otherwise.

2.2 Bézier Curves

A *Bézier curve* is a mapping from the unit interval that is determined by a set of control points $\{\boldsymbol{p}_j\}_{j=0}^n \subset \mathbf{R}^d$. For a parameter $s \in [0, 1]$, there is a corresponding point on the curve:

$$b(s) = \sum_{j=0}^{n} \binom{n}{j} (1 - s)^{n-j} s^j \boldsymbol{p}_j \in \mathbf{R}^d. \tag{1}$$

This is a combination of the control points weighted by each Bernstein basis function $B_{j,n}(s) = \binom{n}{j}(1 - s)^{n-j} s^j$. Due to the binomial expansion $1 = (s + (1 - s))^n = \sum_{j=0}^n B_{j,n}(s)$, a Bernstein basis function is in $[0, 1]$ when s is as well. Due to this fact, the curve must be contained in the convex hull of it's control points.

2.3 Bézier Triangles

A *Bézier triangle* ([Far01, Chapter 17]) is a mapping from the unit triangle $\mathcal{U}$ and is determined by a control net $\{\boldsymbol{p}_{i,j,k}\}_{i+j+k=n} \subset \mathbf{R}^d$. A Bézier triangle is a particular kind of Bézier surface, i.e. one in which there are two cartesian or three barycentric input parameters. Often the term Bézier surface is used to refer to a tensor product or rectangular patch. For $(s, t) \in \mathcal{U}$ we can define barycentric weights $\lambda_1 = 1 - s - t, \lambda_2 = s, \lambda_3 = t$ so that

$$1 = (\lambda_1 + \lambda_2 + \lambda_3)^n = \sum_{\substack{i+j+k=n \\ i,j,k \geq 0}} \binom{n}{i, j, k} \lambda_1^i \lambda_2^j \lambda_3^k. \tag{2}$$

[1] I.e. the degree of the discrete field on the mesh is same as the degree of the shape functions that determine the mesh.

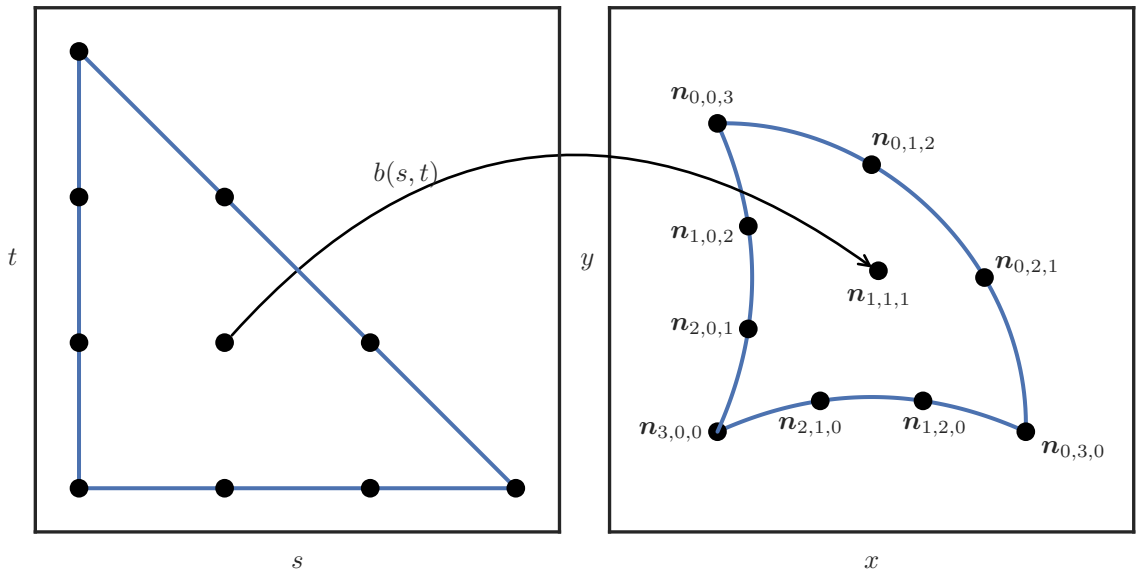

Figure 1: Cubic Bézier triangle

Using this we can similarly define a (triangular) Bernstein basis

$$B_{i,j,k}(s,t) = \binom{n}{i,j,k}(1-s-t)^i s^j t^k = \binom{n}{i,j,k}\lambda_1^i\lambda_2^j\lambda_3^k \tag{3}$$

that is in $[0,1]$ when (s,t) is in $\mathcal{U}$. Using this, we define points on the Bézier triangle as a convex combination of the control net:

$$b(s,t) = \sum_{i+j+k=n}\binom{n}{i,j,k}\lambda_1^i\lambda_2^j\lambda_3^k\boldsymbol{p}_{i,j,k} \in \mathbf{R}^d. \tag{4}$$

Rather than defining a Bézier triangle by the control net, it can also be uniquely determined by the image of a standard lattice of points in $\mathcal{U}$: $b\left(j/n, k/n\right) = \boldsymbol{n}_{i,j,k}$; we'll refer to these as *standard nodes*. Figure 1 shows these standard nodes for a cubic triangle in $\mathbf{R}^2$. To see the correspondence, when $p = 1$ the standard nodes *are* the control net

$$b(s,t) = \lambda_1\boldsymbol{n}_{1,0,0} + \lambda_2\boldsymbol{n}_{0,1,0} + \lambda_3\boldsymbol{n}_{0,0,1} \tag{5}$$

and when $p = 2$

$$\begin{aligned} b(s,t) = \lambda_1\left(2\lambda_1 - 1\right)\boldsymbol{n}_{2,0,0} + \lambda_2\left(2\lambda_2 - 1\right)\boldsymbol{n}_{0,2,0} + \lambda_3\left(2\lambda_3 - 1\right)\boldsymbol{n}_{0,0,2} + \\ 4\lambda_1\lambda_2\boldsymbol{n}_{1,1,0} + 4\lambda_2\lambda_3\boldsymbol{n}_{0,1,1} + 4\lambda_3\lambda_1\boldsymbol{n}_{1,0,1}. \end{aligned} \tag{6}$$

However, it's worth noting that the transformation between the control net and the standard nodes has condition number that grows exponentially with n (see [Far91], which is related but does not directly show this). This may make working with higher degree triangles prohibitively unstable.

A *valid* Bézier triangle is one which is diffeomorphic to $\mathcal{U}$, i.e. $b(s,t)$ is bijective and has an everywhere invertible Jacobian. We must also have the orientation preserved, i.e. the Jacobian must have positive determinant. For example, in Figure 2, the image of $\mathcal{U}$ under the map $b(s,t) = \begin{bmatrix} (1-s-t)^2 + s^2 & s^2 + t^2 \end{bmatrix}^T$ is not valid because the Jacobian is zero along the curve $s^2 - st - t^2 - s + t = 0$ (the dashed line). Elements that are not valid are called *inverted* because they have regions with "negative area". For the example, the image $b\left(\mathcal{U}\right)$ leaves the boundary determined by the edge curves: $b(r,0)$, $b(1-r,r)$ and $b(0,1-r)$ when $r \in [0,1]$. This region outside the boundary is traced twice, once with a positive Jacobian and once with a negative Jacobian.

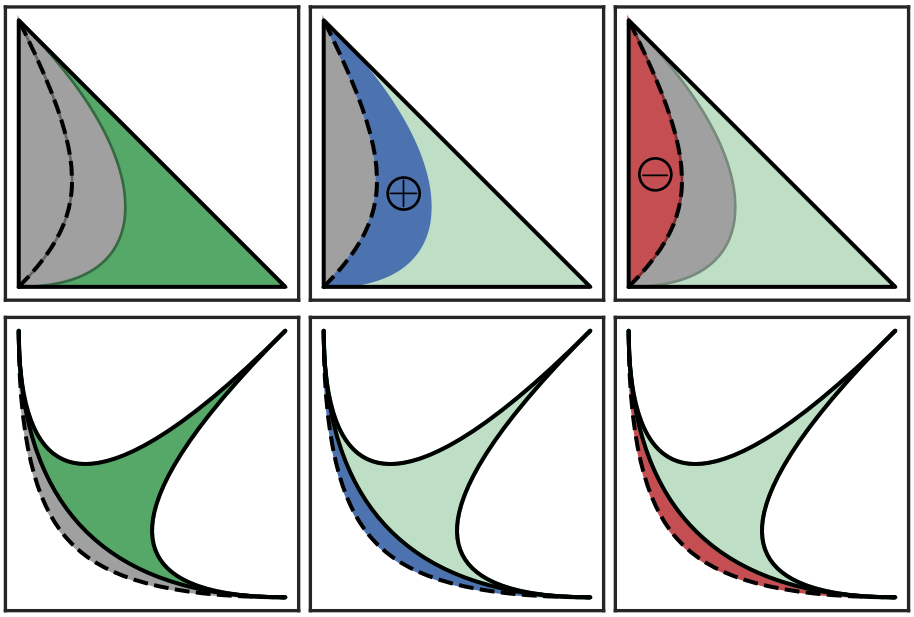

Figure 2: The Bézier triangle given by $b(s,t) = \left[(1-s-t)^2 + s^2 \;\; s^2 + t^2\right]^T$ produces an inverted element. It traces the same region twice, once with a positive Jacobian (the middle column) and once with a negative Jacobian (the right column).

2.4 Curved Elements

We define a curved mesh element $\mathcal{T}$ of degree p to be a Bézier triangle in $\mathbf{R}^2$ of the same degree. We refer to the component functions of $b(s,t)$ (the map that gives $\mathcal{T} = b(\mathcal{U})$) as $x(s,t)$ and $y(s,t)$.

This fits a typical definition ([JM09, Chapter 12]) of a curved element, but gives a special meaning to the mapping from the reference triangle. Interpreting elements as Bézier triangles has been used for Lagrangian methods where mesh adaptivity is needed (e.g. [CMOP04]). Typically curved elements only have one curved side ([MM72]) since they are used to resolve geometric features of a boundary. See also [Zlá73, Zlá74]. Bézier curves and triangles have a number of mathematical properties (e.g. the convex hull property) that lead to elegant geometric descriptions and algorithms.

Note that a Bézier triangle can be determined from many different sources of data (for example the control net or the standard nodes). The choice of this data may be changed to suit the underlying physical problem without changing the actual mapping. Conversely, the data can be fixed (e.g. as the control net) to avoid costly basis conversion; once fixed, the equations of motion and other PDE terms can be recast relative to the new basis (for an example, see [PBP09], where the domain varies with time but the problem is reduced to solving a transformed conservation law in a fixed reference configuration).

2.5 Shape Functions

When defining shape functions (i.e. a basis with geometric meaning) on a curved element there are (at least) two choices. When the degree of the shape functions is the same as the degree of the function being represented on the Bézier triangle, we say the element $\mathcal{T}$ is *isoparametric*. For the multi-index $\boldsymbol{i} = (i,j,k)$, we define $\boldsymbol{u_i} = (j/n, k/n)$ and the corresponding standard node $\boldsymbol{n_i} = b(\boldsymbol{u_i})$. Given these points, two choices for shape functions present themselves:

- *Pre-Image Basis*: $\phi_{\boldsymbol{j}}(\boldsymbol{n_i}) = \widehat{\phi}_{\boldsymbol{j}}(\boldsymbol{u_i}) = \widehat{\phi}_{\boldsymbol{j}}\left(b^{-1}(\boldsymbol{n_i})\right)$ where $\widehat{\phi}_{\boldsymbol{j}}$ is a canonical basis function on $\mathcal{U}$, i.e. $\widehat{\phi}_{\boldsymbol{j}}$ a degree p bivariate polynomial and $\widehat{\phi}_{\boldsymbol{j}}(\boldsymbol{u_i}) = \delta_{\boldsymbol{ij}}$
- *Global Coordinates Basis*: $\phi_{\boldsymbol{j}}(\boldsymbol{n_i}) = \delta_{\boldsymbol{ij}}$, i.e. a canonical basis function on the standard nodes $\{\boldsymbol{n_i}\}$.

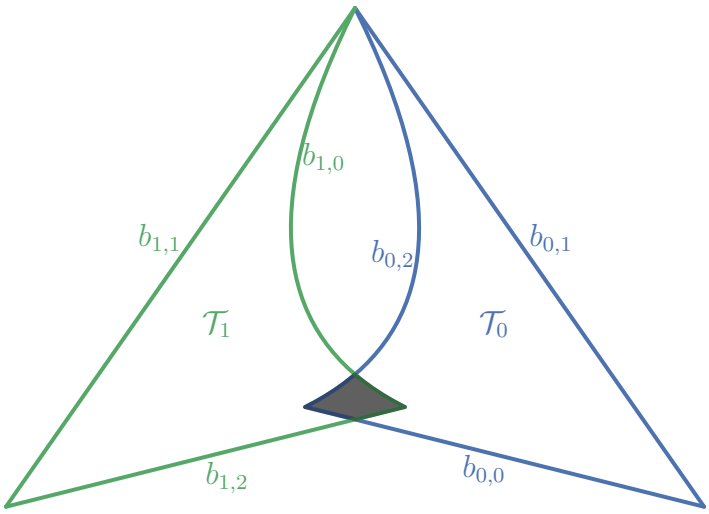

Figure 3: Intersection of Bézier triangles form a curved polygon.

For example, consider a quadratic Bézier triangle:

$$b(s,t) = \begin{bmatrix} 4(st+s+t) & 4(st+t+1) \end{bmatrix}^T \tag{7}$$

$$\implies \begin{bmatrix} \boldsymbol{n}_{2,0,0} & \boldsymbol{n}_{1,1,0} & \boldsymbol{n}_{0,2,0} & \boldsymbol{n}_{1,0,1} & \boldsymbol{n}_{0,1,1} & \boldsymbol{n}_{0,0,2} \end{bmatrix} = \begin{bmatrix} 0 & 2 & 4 & 2 & 5 & 4 \\ 4 & 4 & 4 & 6 & 7 & 8 \end{bmatrix}. \tag{8}$$

In the *Global Coordinates Basis*, we have

$$\phi^G_{0,1,1}(x,y) = \frac{(y-4)(x-y+4)}{6}. \tag{9}$$

For the *Pre-Image Basis*, we need the inverse and the canonical basis

$$b^{-1}(x,y) = \begin{bmatrix} \frac{x-y+4}{4} & \frac{y-4}{x-y+8} \end{bmatrix} \quad \text{and} \quad \widehat{\phi}_{0,1,1}(s,t) = 4st \tag{10}$$

and together they give

$$\phi^P_{0,1,1}(x,y) = \frac{(y-4)(x-y+4)}{x-y+8}. \tag{11}$$

In general $\phi^P_{\boldsymbol{j}}$ may not even be a rational bivariate function; due to composition with b^{-1} we can only guarantee that it is algebraic (i.e. it can be defined as the zero set of polynomials).

2.6 Curved Polygons

When intersecting two curved elements, the resulting surface(s) will be defined by the boundary, alternating between edges of each element. For example, in Figure 3, a "curved quadrilateral" is formed when two Bézier triangles $\mathcal{T}_0$ and $\mathcal{T}_1$ are intersected.

A *curved polygon* is defined by a collection of Bézier curves in $\mathbf{R}^2$ that determine the boundary. In order to be a valid polygon, none of the boundary curves may cross, the ends of consecutive edge curves must meet and the curves must be right-hand oriented. For our example in Figure 3, the triangles have boundaries formed by three Bézier curves: $\partial\mathcal{T}_0 = b_{0,0} \cup b_{0,1} \cup b_{0,2}$ and $\partial\mathcal{T}_1 = b_{1,0} \cup b_{1,1} \cup b_{1,2}$. The intersection $\mathcal{P}$ is defined by four boundary curves: $\partial\mathcal{P} = C_1 \cup C_2 \cup C_3 \cup C_4$. Each boundary curve is itself a Bézier curve[2]: $C_1 = b_{0,0}([0, 1/8])$, $C_2 = b_{1,2}([7/8, 1])$, $C_3 = b_{1,0}([0, 1/7])$ and $C_4 = b_{0,2}([6/7, 1])$.

Though an intersection can be described in terms of the Bézier triangles, the structure of the control net will be lost. The region will not in general be able to be described by a mapping from a simple space like $\mathcal{U}$.

[2] A specialization of a Bézier curve $b([a_1, a_2])$ is also a Bézier curve.

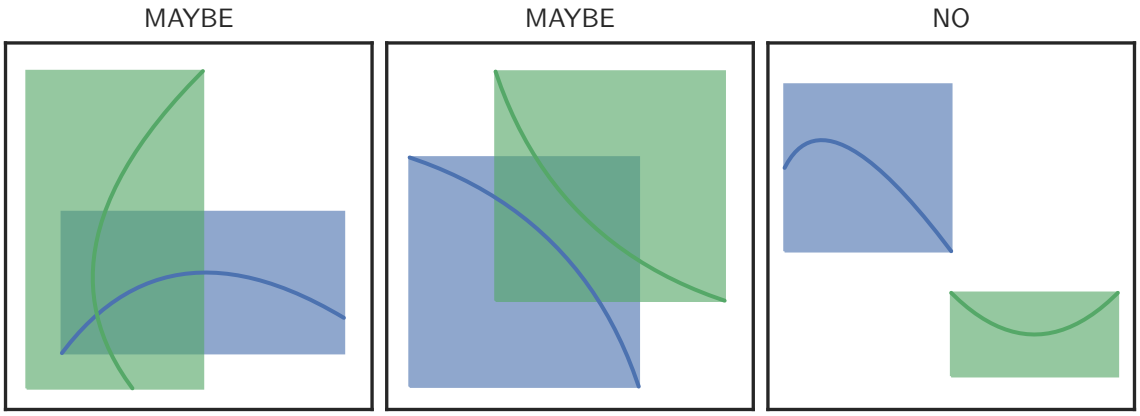

Figure 4: Bounding box intersection predicate. This is a cheap way to conclude that two curves don't intersect, though it inherently is susceptible to false positives.

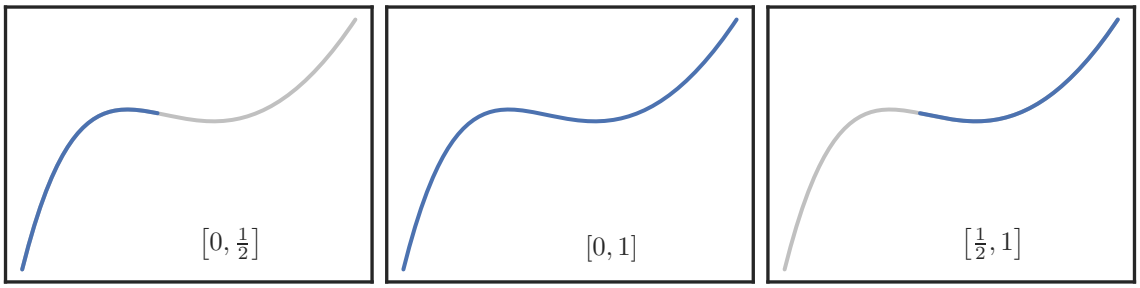

Figure 5: Bézier curve subdivision.

3 Bézier Intersection Problems

3.1 Intersecting Bézier Curves

The problem of intersecting two Bézier curves is a core building block for intersecting two Bézier triangles in $\mathbf{R}^2$ Since a curve is an algebraic variety of dimension one, the intersections will either be a curve segment common to both curves (if they coincide) or a finite set of points (i.e. dimension zero). Many algorithms have been described in the literature, both geometric ([SP86, SN90, KLS98]) and algebraic ([MD92]).

In the implementation for this work, the Bézier subdivision algorithm is used. In the case of a transversal intersection (i.e. one where the tangents to each curve are not parallel and both are non-zero), this algorithm performs very well. However, when curves are tangent, a large number of (false) candidate intersections are detected and convergence of Newton's method slows once in a neighborhood of an actual intersection. Non-transversal intersections have infinite condition number, but transversal intersections with very high condition number can also cause convergence problems.

In the Bézier subdivision algorithm, we first check if the bounding boxes for the curves are disjoint (Figure 4). We use the bounding boxes rather than the convex hulls since they are easier to compute and the intersections of boxes are easier to check. If they are disjoint, the pair can be rejected. If not, each curve $\mathcal{C} = b([0, 1])$ is split into two halves by splitting the unit interval: $b\left(\left[0, \frac{1}{2}\right]\right)$ and $b\left(\left[\frac{1}{2}, 1\right]\right)$ (Figure 5).

As the subdivision continues, some pairs of curve segments may be kept around that won't lead to an intersection (Figure 6). Once the curve segments are close to linear within a given tolerance (Figure 7), the process terminates.

Once both curve segments are linear (to tolerance), the intersection is approximated by intersecting the lines connecting the endpoints of each curve segment. This approximation is used as a starting point for Newton's method, to find a root of $F(s, t) = b_0(s) - b_1(t)$. Since $b_0(s), b_1(t) \in \mathbf{R}^2$ we have Jacobian

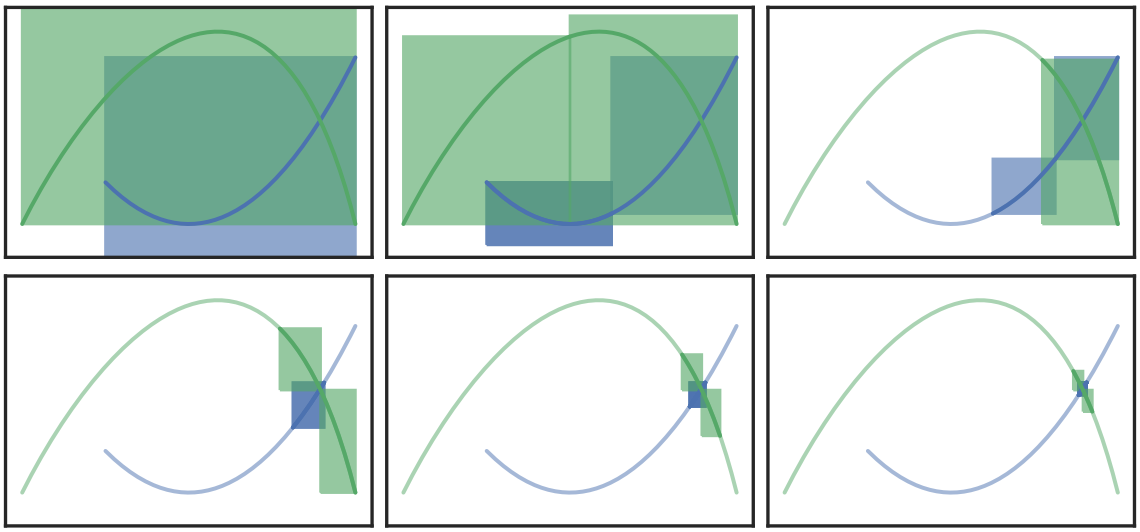

Figure 6: Bézier subdivision algorithm.

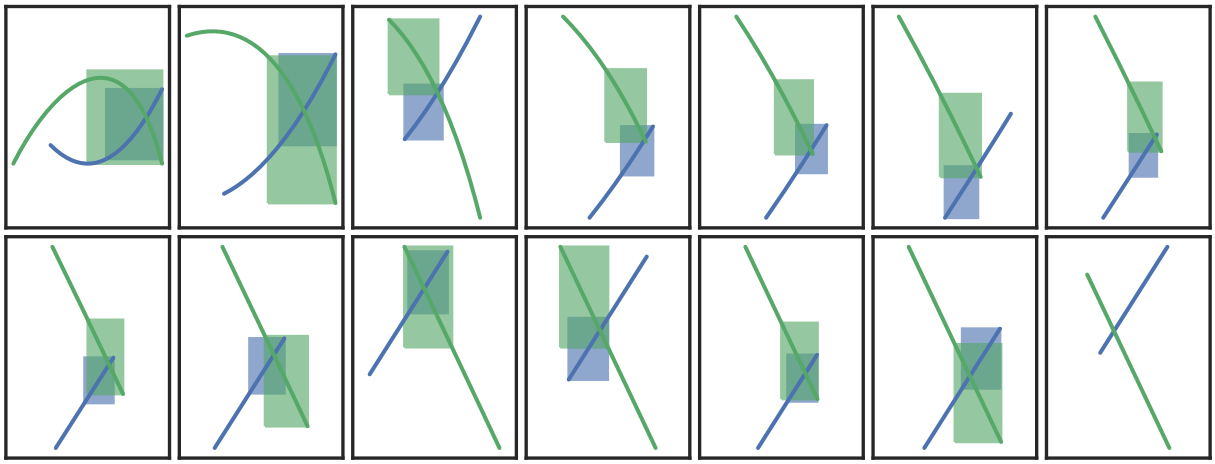

Figure 7: Subdividing until linear within tolerance.

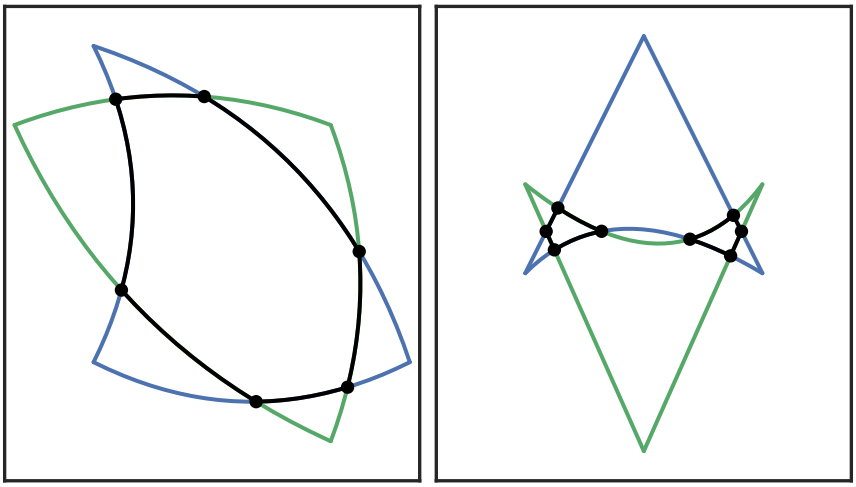

Figure 8: Edge intersections during Bézier triangle intersection.

$J = \begin{bmatrix} b_0'(s) & -b_1'(t) \end{bmatrix}$. With these, Newton's method is

$$\begin{bmatrix} s_{n+1} & t_{n+1} \end{bmatrix}^T = \begin{bmatrix} s_n & t_n \end{bmatrix}^T - J_n^{-1} F_n. \tag{12}$$

This also gives an indication why convergence issues occur at non-transveral intersections: they are exactly the intersections where the Jacobian is singular.

3.2 Intersecting Bézier Triangles

The chief difficulty in intersecting two surfaces is intersecting their edges, which are Bézier curves. Though this is just a part of the overall algorithm, it proved to be the *most difficult* to implement ([Her17]). So the first part of the algorithm is to find all points where the edges intersect (Figure 8).

To determine the curve segments that bound the curved polygon region(s) (see Section 2.6 for more about curved polygons) of intersection, we not only need to keep track of the coordinates of intersection, we also need to keep note of *which* edges the intersection occurred on and the parameters along each curve. With this information, we can classify each point of intersection according to which of the two curves forms the boundary of the curved polygon (Figure 9). Using the right-hand rule we can compare the tangent vectors on each curve to determine which one is on the interior.

This classification becomes more difficult when the curves are tangent at an intersection, when the intersection occurs at a corner of one of the surfaces or when two intersecting edges are coincident on the same algebraic curve (Figure 10).

In the case of tangency, the intersection is non-transversal, hence has infinite condition number. In the case of coincident curves, there are infinitely many intersections (along the segment when the curves coincide) so the subdivision process breaks down.

3.2.1 Example

Consider two Bézier surfaces (Figure 11)

$$b_0(s,t) = \begin{bmatrix} 8s \\ 8t \end{bmatrix} \qquad b_1(s,t) = \begin{bmatrix} 2(6s+t-1) \\ 2(8s^2+8st-8s+3t+2) \end{bmatrix} \tag{13}$$

In the *first step* we find all intersections of the edge curves

$$E_0(r) = \begin{bmatrix} 8r \\ 0 \end{bmatrix}, E_1(r) = \begin{bmatrix} 8(1-r) \\ 8r \end{bmatrix}, E_2(r) = \begin{bmatrix} 0 \\ 8(1-r) \end{bmatrix},$$

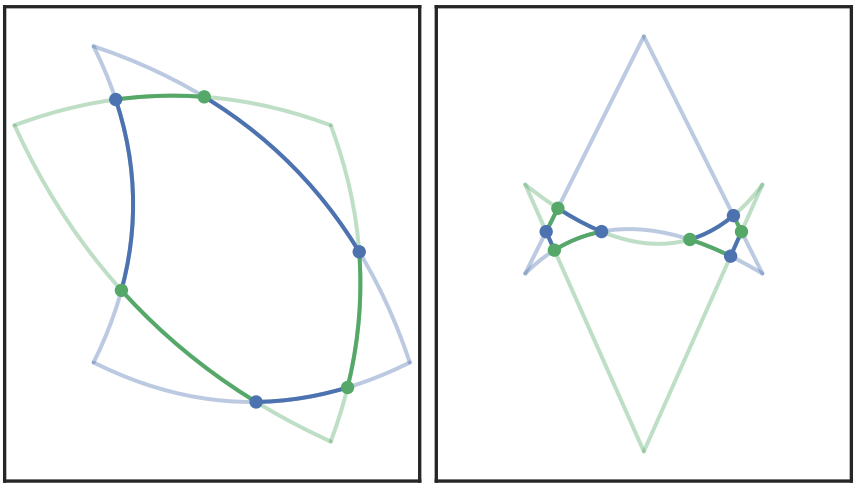

Figure 9: Classified intersections during Bézier triangle intersection.

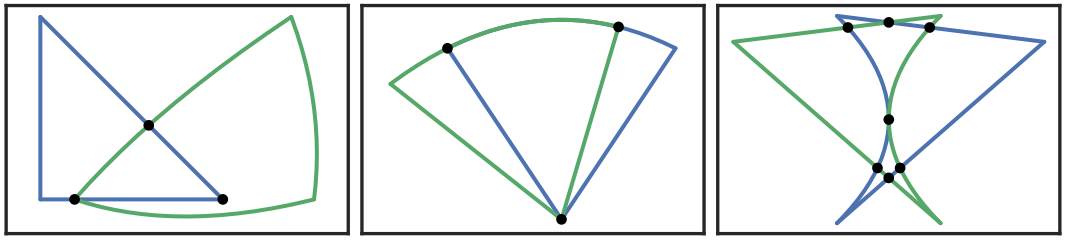

Figure 10: Bézier triangle intersection difficulties.

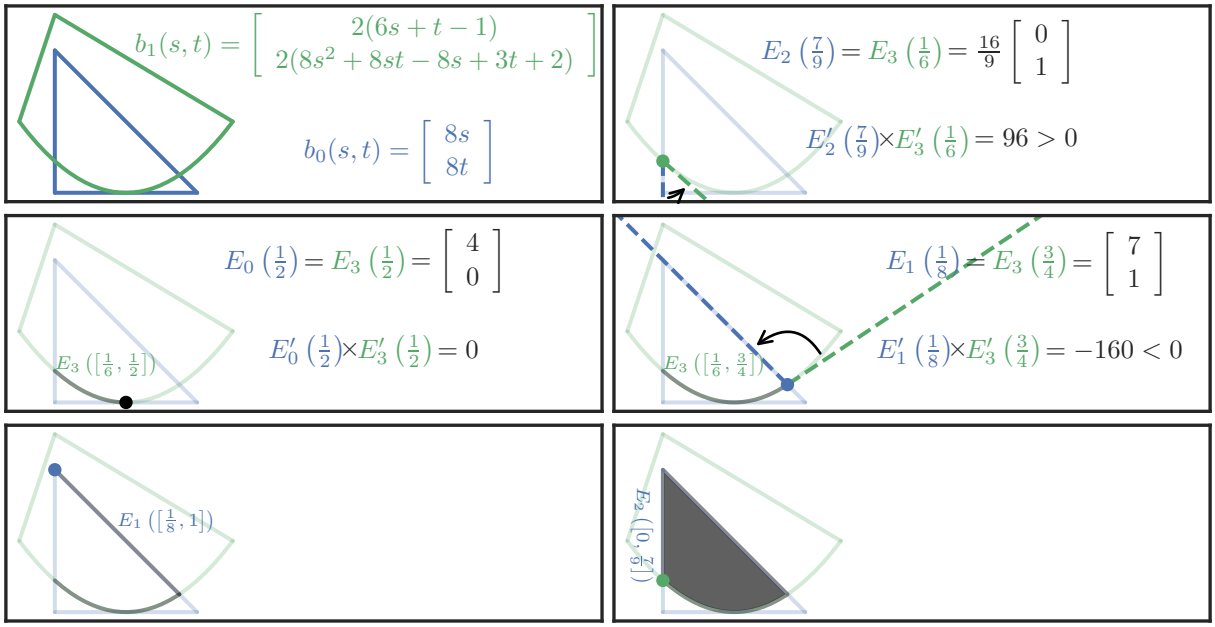

Figure 11: Surface Intersection Example

$$E_3(r) = \begin{bmatrix} 2(6r-1) \\ 4(2r-1)^2 \end{bmatrix}, E_4(r) = \begin{bmatrix} 10(1-r) \\ 2(3r+2) \end{bmatrix}, E_5(r) = \begin{bmatrix} -2r \\ 2(5-3r) \end{bmatrix}. \quad (14)$$

We find three intersections and we classify each of them by comparing the tangent vectors

$$I_1 : E_2\left(\frac{7}{9}\right) = E_3\left(\frac{1}{6}\right) = \frac{16}{9}\begin{bmatrix} 0 \\ 1 \end{bmatrix} \Longrightarrow E_2'\left(\frac{7}{9}\right) \times E_3'\left(\frac{1}{6}\right) = 96 \quad (15)$$

$$I_2 : E_0\left(\frac{1}{2}\right) = E_3\left(\frac{1}{2}\right) = \begin{bmatrix} 4 \\ 0 \end{bmatrix} \Longrightarrow E_0'\left(\frac{1}{2}\right) \times E_3'\left(\frac{1}{2}\right) = 0 \quad (16)$$

$$I_3 : E_1\left(\frac{1}{8}\right) = E_3\left(\frac{3}{4}\right) = \begin{bmatrix} 7 \\ 1 \end{bmatrix} \Longrightarrow E_1'\left(\frac{1}{8}\right) \times E_3'\left(\frac{3}{4}\right) = -160. \quad (17)$$

From here, we construct our curved polygon intersection by drawing from our list of intersections until none remain.

- First consider I_1. Since $E_2' \times E_3' > 0$ at this point, then we consider the curve E_3 to be *interior*.
- After classification, we move along E_3 until we encounter another intersection: I_2
- I_2 is a point of tangency since $E_0'\left(\frac{1}{2}\right) \times E_3'\left(\frac{1}{2}\right) = 0$. Since a tangency has no impact on the underlying intersection geometry, we ignore it and keep moving.
- Continuing to move along E_3, we encounter another intersection: I_3. Since $E_1' \times E_3' < 0$ at this point, we consider the curve E_1 to be *interior* at the intersection. Thus we stop moving along E_3 and we have our first curved segment: $E_3\left(\left[\frac{1}{6}, \frac{3}{4}\right]\right)$
- Finding no other intersections on E_1 we continue until the end of the edge. Now our (ordered) curved segments are:

$$E_3\left(\left[\frac{1}{6}, \frac{3}{4}\right]\right) \longrightarrow E_1\left(\left[\frac{1}{8}, 1\right]\right). \quad (18)$$

- Next we stay at the corner and switch to the next curve E_2, moving along that curve until we hit the next intersecton I_1. Now our (ordered) curved segments are:

$$E_3\left(\left[\frac{1}{6}, \frac{3}{4}\right]\right) \longrightarrow E_1\left(\left[\frac{1}{8}, 1\right]\right) \longrightarrow E_2\left(\left[0, \frac{7}{9}\right]\right). \quad (19)$$

  Since we are now back where we started (at I_1) the process stops

We represent the boundary of the curved polygon as Bézier curves, so to complete the process we reparameterize ([Far01, Ch. 5.4]) each curve onto the relevant interval. For example, E_3 has control points $p_0 = \begin{bmatrix} -2 \\ 4 \end{bmatrix}$, $p_1 = \begin{bmatrix} 4 \\ -4 \end{bmatrix}$, $p_2 = \begin{bmatrix} 10 \\ 4 \end{bmatrix}$ and we reparameterize on $\alpha = \frac{1}{6}, \beta = \frac{3}{4}$ to control points

$$q_0 = E_3\left(\frac{1}{6}\right) = \frac{16}{9}\begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad (20)$$

$$q_1 = (1-\alpha)\left[(1-\beta)p_0 + \beta p_1\right] + \alpha\left[(1-\beta)p_1 + \beta p_2\right] = \frac{1}{6}\begin{bmatrix} 21 \\ -8 \end{bmatrix} \quad (21)$$

$$q_2 = E_3\left(\frac{3}{4}\right) = \begin{bmatrix} 7 \\ 1 \end{bmatrix}. \quad (22)$$

3.3 Bézier Triangle Inverse

The problem of determining the parameters (s, t) given a point $\boldsymbol{p} = \begin{bmatrix} x & y \end{bmatrix}^T$ in a Bézier triangle can also be solved by using subdivision with a bounding box predicate and then Newton's method at the end.

For example, Figure 12 shows how regions of $\mathcal{U}$ can be discarded recursively until the suitable region for (s, t) has a sufficiently small area. At this point, we can apply Newton's method to the map $F(s, t) = b(s, t) - \boldsymbol{p}$. It's very helpful (for Newton's method) that $F : \mathbf{R}^2 \longrightarrow \mathbf{R}^2$ since the Jacobian will always be invertible when the Bézier triangle is valid. If $\boldsymbol{p} \in \mathbf{R}^3$ then the system would be underdetermined. Similarly, if $\boldsymbol{p} \in \mathbf{R}^2$ but $b(s)$ is a Bézier curve then the system would be overdetermined.

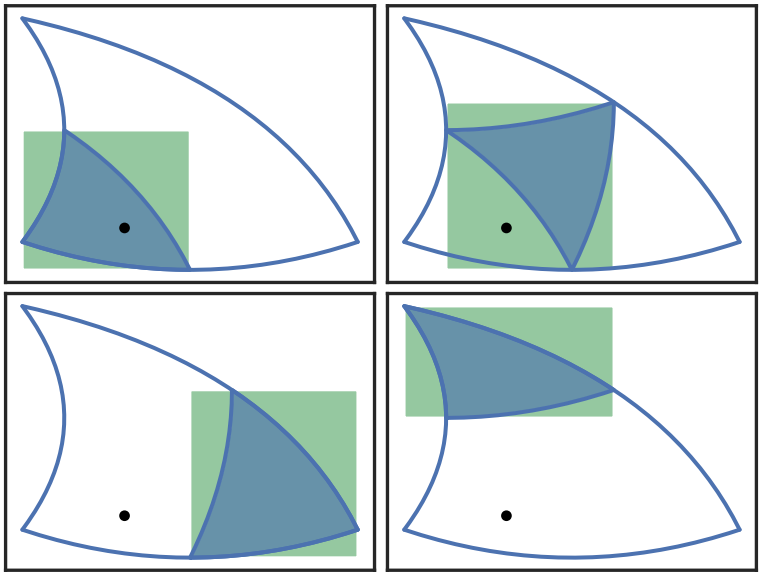

Figure 12: Checking for a point $\boldsymbol{p}$ in each of four subregions when subdividing a Bézier triangle.

4 Solution Transfer

Placeholder.

References

[BR78] I. Babuška and W. C. Rheinboldt. Error Estimates for Adaptive Finite Element Computations. *SIAM Journal on Numerical Analysis*, 15(4):736–754, 1978.

[CMOP04] David E. Cardoze, Gary L. Miller, Mark Olah, and Todd Phillips. A Bézier-Based Moving Mesh Framework for Simulation with Elastic Membranes. In *Proceedings of the 13th International Meshing Roundtable, IMR 2004, Williamsburg, Virginia, USA, September 19-22, 2004*, pages 71–80, 2004.

[Far91] R.T. Farouki. On the stability of transformations between power and Bernstein polynomial forms. *Computer Aided Geometric Design*, 8(1):29–36, Feb 1991.

[Far01] Gerald Farin. *Curves and Surfaces for CAGD, Fifth Edition: A Practical Guide (The Morgan Kaufmann Series in Computer Graphics)*. Morgan Kaufmann, 2001.

[FM11] P.E. Farrell and J.R. Maddison. Conservative interpolation between volume meshes by local Galerkin projection. *Computer Methods in Applied Mechanics and Engineering*, 200(1-4):89–100, Jan 2011.

[FPP+09] P.E. Farrell, M.D. Piggott, C.C. Pain, G.J. Gorman, and C.R. Wilson. Conservative interpolation between unstructured meshes via supermesh construction. *Computer Methods in Applied Mechanics and Engineering*, 198(33-36):2632–2642, Jul 2009.

[HAC74] C.W Hirt, A.A Amsden, and J.L Cook. An arbitrary Lagrangian-Eulerian computing method for all flow speeds. *Journal of Computational Physics*, 14(3):227–253, Mar 1974.

[Her17] Danny Hermes. Helper for Bézier Curves, Triangles, and Higher Order Objects. *The Journal of Open Source Software*, 2(16):267, Aug 2017.

[JH04] Xiangmin Jiao and Michael T. Heath. Common-refinement-based data transfer between non-matching meshes in multiphysics simulations. *International Journal for Numerical Methods in Engineering*, 61(14):2402–2427, 2004.

[JM09] Claes Johnson and Mathematics. *Numerical Solution of Partial Differential Equations by the Finite Element Method (Dover Books on Mathematics)*. Dover Publications, 2009.

[KLS98] Deok-Soo Kim, Soon-Woong Lee, and Hayong Shin. A cocktail algorithm for planar Bézier curve intersections. *Computer-Aided Design*, 30(13):1047–1051, Nov 1998.

[MD92] Dinesh Manocha and James W. Demmel. Algorithms for Intersecting Parametric and Algebraic Curves. Technical Report UCB/CSD-92-698, EECS Department, University of California, Berkeley, Aug 1992.

[MM72] R. McLeod and A. R. Mitchell. The Construction of Basis Functions for Curved Elements in the Finite Element Method. *IMA Journal of Applied Mathematics*, 10(3):382–393, 1972.

[PBP09] P.-O. Persson, J. Bonet, and J. Peraire. Discontinuous Galerkin solution of the Navier–Stokes equations on deformable domains. *Computer Methods in Applied Mechanics and Engineering*, 198(17-20):1585–1595, Apr 2009.

[PUdOG01] C.C. Pain, A.P. Umpleby, C.R.E. de Oliveira, and A.J.H. Goddard. Tetrahedral mesh optimisation and adaptivity for steady-state and transient finite element calculations. *Computer Methods in Applied Mechanics and Engineering*, 190(29-30):3771–3796, Apr 2001.

[PVMZ87] J Peraire, M Vahdati, K Morgan, and O.C Zienkiewicz. Adaptive remeshing for compressible flow computations. *Journal of Computational Physics*, 72(2):449–466, Oct 1987.

[SN90] T.W. Sederberg and T. Nishita. Curve intersection using Bézier clipping. *Computer-Aided Design*, 22(9):538–549, Nov 1990.

[SP86] Thomas W Sederberg and Scott R Parry. Comparison of three curve intersection algorithms. *Computer-Aided Design*, 18(1):58–63, Jan 1986.

[WFA+13] Z.J. Wang, Krzysztof Fidkowski, Rémi Abgrall, Francesco Bassi, Doru Caraeni, Andrew Cary, Herman Deconinck, Ralf Hartmann, Koen Hillewaert, H.T. Huynh, Norbert Kroll, Georg May, Per-Olof Persson, Bram van Leer, and Miguel Visbal. High-order CFD methods: current status and perspective. *International Journal for Numerical Methods in Fluids*, 72(8):811–845, Jan 2013.

[Zlá73] Miloš Zlámal. Curved Elements in the Finite Element Method. I. *SIAM Journal on Numerical Analysis*, 10(1):229–240, Mar 1973.

[Zlá74] Miloš Zlámal. Curved Elements in the Finite Element Method. II. *SIAM Journal on Numerical Analysis*, 11(2):347–362, Apr 1974.